dsm_loader
User Guide

Published by:

Endace Measurement Systems® Ltd

Building 7
17 Lambie Drive

PO Box 76802
Manukau City 1702
New Zealand

Phone: +64 9 262 7260
Fax: +64 9 262 7261

support@endace.com
www.endace.com

International Locations

New Zealand
Endace Technology® Ltd

Level 9
85 Alexandra Street
PO Box 19246
Hamilton 2001
New Zealand

Phone: +64 7 839 0540
Fax: +64 7 839 0543

support@endace.com
www.endace.com

Americas
Endace USA® Ltd

Suite 220
11495 Sunset Hill Road
Reston
Virginia 20190
United States of America

Phone: ++1 703 382 0155
Fax: ++1 703 382 0155

support@endace.com
www.endace.com

Europe, Middle East & Africa
Endace Europe® Ltd

Sheraton House
Castle Park
Cambridge CB3 0AX
United Kingdom

Phone: ++44 1223 370 176
Fax: ++44 1223 370 040

support@endace.com
www.endace.com

Disclaimer
Whilst every effort has been made to ensure accuracy, neither Endace Measurement Systems® Ltd nor any employee of the company, shall be liable on any ground whatsoever to any party in respect of decisions or actions they may make as a result of using the information contained in this document.

In accordance with the Endace Measurement Systems® Ltd policy of continuing development, design and specifications are subject to change without notice.

Table of Contents

1. Purpose

This document describes the usage of the `dsm_loader` application.

1.1 Revision History

Revision	Data of Change	Description of Change	Revision Originator
1.0	27/02/06	Initial revision	Ben Gray

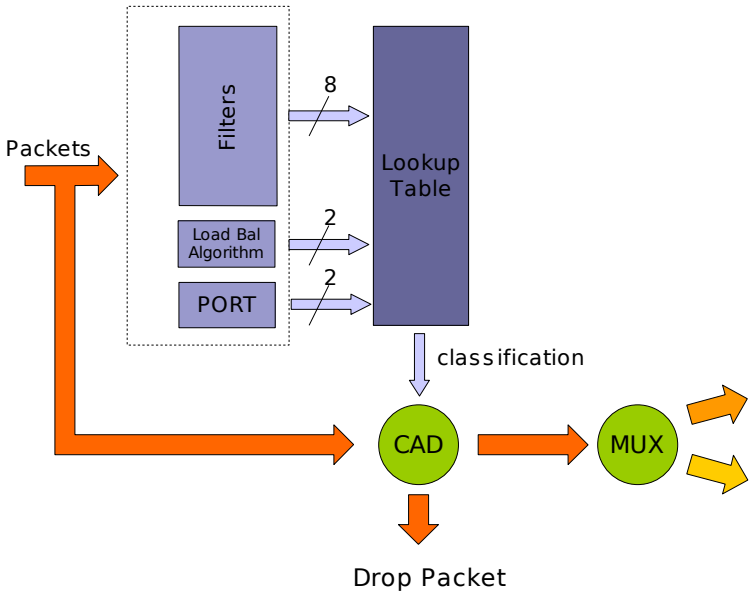
2. Data Stream Manager (DSM) Introduction

2.1 DSM Overview

The Data Stream Manager (DSM) is a feature supported on DAG4.5 and DAG6.2 cards, it provides functionality to drop or route packets to a particular receive stream based on the packet contents, physical port and the output of two load balancing algorithms. The DSM logic is implement in firmware on the DAG card, it does not require host CPU intervention once configured.

Figure 2-1 shows the logical flow of packet records in the DSM module.

Figure 2-1. Data Stream Manager (DSM) block diagram



Packets are received from the line and stamped with an ERF (Endace Record Format) header, then past along to the filter and load balancing block.

- **Filter / Load Balancing Block**

  The filter block applies eight bit-mask filters simultaneously to the start of the packet, producing a single true/false value for each filter. The load balancing[1] block applies two algorithms to the packet data, again producing one true/false boolean output per algorithm.

1 Load balancing(LB) is also known as Hash Load Balancing (HLB) or simply as just steering algorithm.

- **Lookup Table Block**

  Accepting the filter and load balancing outputs is the lookup table, it also receives the physical port the packet arrived on and calculates a classification for the packet. The classification is also known as the color of the packet.

- **Coloriser and Drop Block**

  The color is then past onto the Coloriser And Drop (CAD) block that checks if the packet should be dropped, if not the color is inserted into the ERF record header of the packet and then the packet record is past along to the packet record multiplexier.

- **Packet Record Multiplexier (ERF MUX)**

  The ERF MUX looks at the color information contained in the packet record and determines which receive stream the packet record should be routed to.

2.1.1 Packet Filters

Prior to packets being present to the DSM module, they are stamped with a ERF record header and possibly snapped to a particular length (set by the `'snap length'` card configuration option). This is standard DAG card behaviour, but should be taken into account when using the DSM firmware as it could effect filter output.

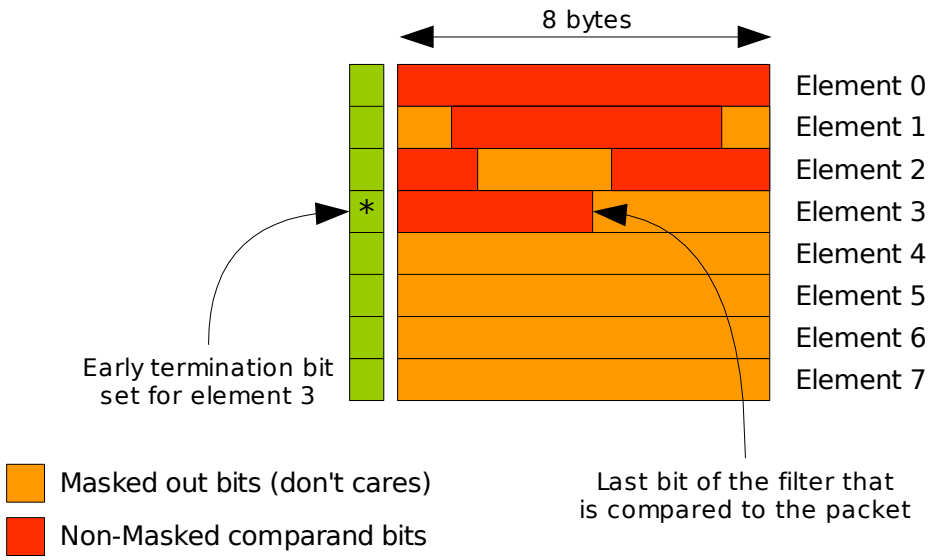
There are eight 64-byte bitmasked filters inside the DSM module, each are compared against the packet in parallel. The first byte of the filter is compared against the first byte of the packet record after the ERF header, refer to the *Endace Extensible Record Format* document for more information on the packet record format. It is important to note that for ethernet packets there are two bytes of padding added immediately after the ERF header, these padding bytes are the first to be compared against the filter.

Each filter outputs a boolean `true` or `false` value that is provided to the lookup table for further classification.

Filters also have an early termination option, this allows the user to specify on which 8-byte chunk (known as an element) of the filter contains the last byte to check. The early termination option is always specified on the last element in the filter (element 7). Packets that are smaller than the filter, as defined by the early termination option, always produce a `false` output regardless of the packet contents.

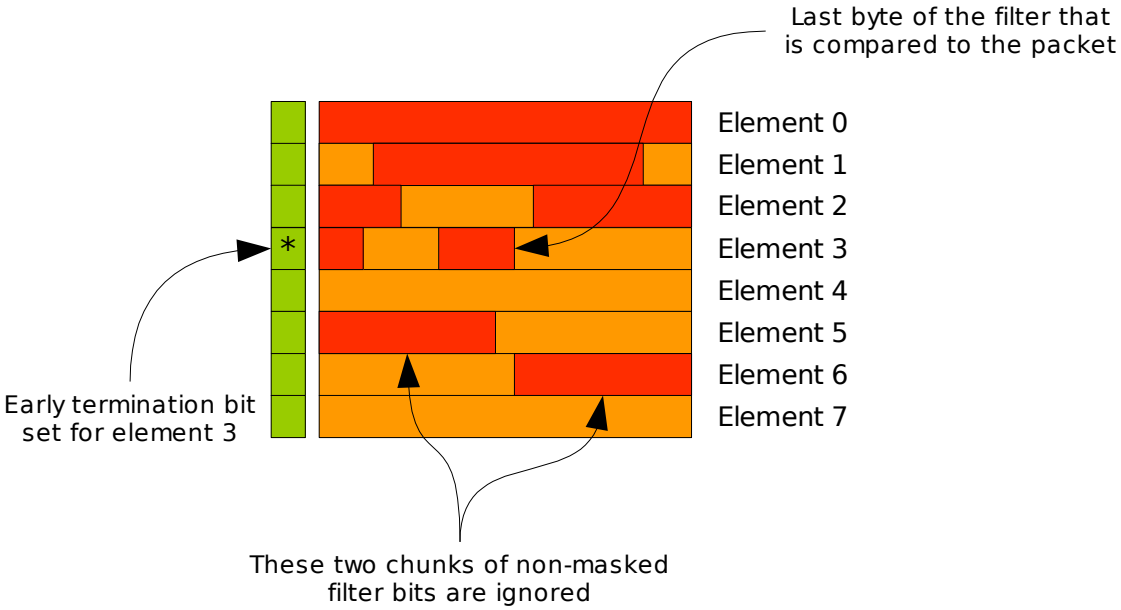
Figure 2-2 shows a logical drawing of a filter, each of the rows represents 8 bytes of the filter (one element). In the diagram, the filter will be applied to the first 28 bytes (3 elements × 8 bytes + 4 non-masked bytes of element 3) of the packets rather than the full 64. Packets that are smaller than 28 bytes will produce a `false` output.

Figure 2-2. Filter example with early termination



The diagram in figure 2-3 shows that any non-masked bytes of the filter that occur in elements after the early termination option are effectively ignored regardless of the packet length.

Figure 2-3. Filter example with ignored non-masked bytes



2.1.2 Load Balancing (Steering) Algorithms

Two load balancing algorithms are applied to the packet, each resulting in a boolean output value, both outputs are provided to the lookup table for further classification. The first algorithm is a CRC calculation applied

to the expected location of an IPv4 packet's source and destination address within the packet record. The second algorithm calculates the parity, across the expected location of an IPv4 packet's source and destination addresses.

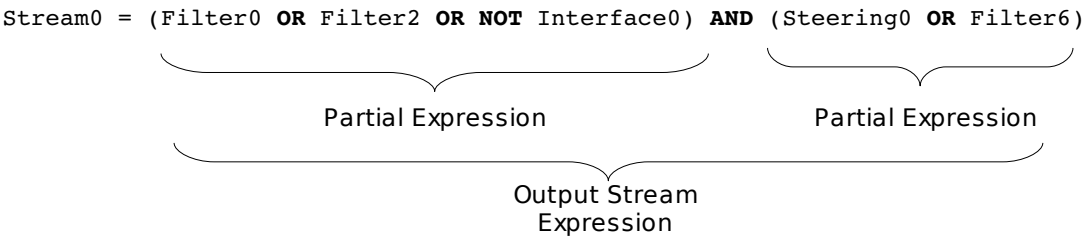
For a random collection of packet data, both algorithms give an approximately 50:50 split of `true` and `false` outputs. The load balancing algorithms are fully implemented in firmware and are not user configurable.

2.1.3 Lookup Table

The lookup table accepts the outputs from the filters, load balancing algorithm and the physical port number of the packet, to generate either a target stream number for the packet or a drop indication. The lookup table is fully user programmable, allowing for complex expressions to be constructed.

The DSM API provides a two stage implementation of the lookup table construction. The first stage involves creating one or more partial expressions, each parameter of the expression(or the inverse of the parameter) is logically OR'ed together to produce the partial expression. In the second stag, stream output expressions are constructed, containing one or more partial expressions, each partial expression (or the inverse of the partial expression) is AND'ed together.

Figure 2-4. Lookup Table Expression



Packet records can be routed to only one stream, if more than one output expression returns a boolean `true` value for a set of input parameters, the stream with the highest priority (lowest stream number) will receive the packet record. For example if the output stream expressions were the same for both stream 0 and stream 2, packet records that are accepted by the expression will only be routed to stream 0.

2.1.4 Output Record Format

Packets that are sent though the DSM are marked with a *color* value, this value encodes the outputs of the eight filters and two load balancing algorithms, as well as the target receive stream. Refer to Appendix A for the format of ERF record header including the color field.

2.1.5 Counters

The DSM module maintains a minimum of thirteen counters, each counter is 32-bits and wraps back to zero on overflow.

Table 2-1. DSM Counters

Type	*Count*	*Description*
Filter	8	Each filter has a counter indicating how many times the filter has output a `true` result.
Load Balancing	2	Both of the load balancing algorithms have a counter indicating how many `true` results have been generated.
Drop	1	Counts the number of packets that have been dropped.
Stream	n*	Each receive stream has a counter indicating the number of packet records that have been routed to that stream.

** the number of receive stream counters depends on the number of receive streams available on the card, currently this is 2.*

3. Loading Configurations

3.1 Card Initialisation

The DAG card may require configuration prior to loading the DSM settings. The following paragraphs illustrate the typically steps required for configuration.

3.1.1 Load DSM Firmware

By default DAG cards are not shipped with the required DSM firmware loaded into the FPGA, to load the correct firmware perform the following command.

```
dagrom -d n -rvp -f filename.bit
```

Where `filename.bit` is the DSM FPGA image to load, and `n` is the number of the card. Depending on the type of card, it may be required to load the packet processing FPGA as well, refer to your DAG card manual for more information.

DAG 4.5G2/G4 Card

Load the DSM PCI firmware image using the following command:

```
dagrom -vrpy –f xilinx/dag45ge-pcix-dsm.bit
```

DAG 6.2SE Card

Load the DSM PCI firmware image using the following command:

```
dagrom_vrpy –f xilinx/dag62sepcix-erf-dsm.bit
```

Load the DSM PP image using the following command:

```
dagld –x /xilinx/dag62sepp-erf-dsm
```

3.2.2 Configure the Card

Refer to the DAG card user manual for details on how to configure the card for your particular network settings. Usually the DAG card configuration is independent of the DSM configuration, however the `snap length` configuration attribute has a direct bearing on the DSM functionality.

DAG 4.5G2/G4 Card

Initialise the DAG 4.5G2/G4 card using the following comand:

`dagconfig default laser slen=2048 mem=64:0:64:0` (assuming you have 128MB of memory assigned to the card)

Verify the card has link and is initialised correctly using `dagconfig –si`

For a DAG 4.5G2 the output of the `dagconfig –si` command should look as follows:

```
Port Link Plink RFault LOF LOS
A     1     1     0     0   0
B     1     1     0     0   0
```

DAG 6.2SE Card

Initialise the DAG 6.2SE card using the following command:

`dagsix default eth slen=2048 crcstrip` (for Gigabit Ethernet)
`dagsix default pos slen=2048 crcstrip` (for OC192)

Verify the card has link and is initialised correctly using `dagsix –si`

The output of the `dagsix –si` command should look as follows:

```
RAI RLE RPA LOS LOC LOF BER LFT RFT FCS_ERR BAD_PACKET GOOD_PACKET RXF

0    0   0   0   0   0   0   0    0    0          0           8770959 0
0    0   0   0   0   0   0   0    0    0          0           8770191 0
0    0   0   0   0   0   0   0    0    0          0           8770830 0
0    0   0   0   0   0   0   0    0    0          0           8770217 0
```

3.2 `dsm_loader` Overview

The command line `dsm_loader` application is a tool that accepts a DSM configuration file and loads the details into the DSM firmware on supported DAG cards. It utilises the DSM software API library to configure the DSM.

3.3 `dsm_loader` Usage

The following arguments can be used with `dsm_loader`

```
Endace DSM Filter and Expression Loader
(c) 2006 Endace Technology Ltd.

dsm_loader - loads filters and output expressions into DSM equipped DAG
cards
Usage : dsm_loader -d <device> [options] -f <config file>
Options:
    -?,--usage
    -h,--help                   display help (this page)
    -v,--verbose                increase verbosity
    -d,--device <device>        DAG device to use
    -f,--config_file <filename> configuration file for the filters
                                and expressions
    -y,--dont_download          don't download the configuration to the
                                card, if not specified the configuration
                                is always download to the card
    -s,--stats <seconds>        display DSM statistics every <seconds>
    -o,--output_file <filename> output file to dump the constructed
                                filters and lookup table
```

The use of a file containing configuration details is mandatory, the format of the file is given in chapter 4 of this document.

The `-y` *don't download* option is useful if you want to simply parse a configuration file and observe the raw DSM configuration using the `-o` option. The file format used for the output file is detailed in Appendix B.

Statistics can be displayed periodically using the `-s` option, the statistics consist of the counters maintained by the DSM firmware.

4. File Format Specification

The filter loader file is written in xml format, it must satisfy the format by having a single root node `<dsm-config>`. Detailed in the following sections are the possible elements that may be used inside the root element.

4.1 XML Element Specification

4.1.1 <dsm-config>

This must be the root element of the xml document.

Table 4-1. <dsm-config> Attributes

Attribute	*Use*	*Type*	*Default Value*	*Description*
version	Mandatory	float	-	Defines the version of the file, currently the only version is 1.0. This document describes version 1.0 only.

Table 4-2. <dsm-config> Elements

Element	*Occurrences*		*Description*	*Section*
	Min	*Max*		
filter	0	7	Filter element that contains the details of a particular filter.	Section 4.1.2
interface	0	Unlimited	Interface element that defines a particular interface and labels it.	Section 4.1.3
steering	0	Unlimited	Steering element that labels a hash load balancing algorithm used by the card.	Section 4.1.4
partial	0	Unlimited	Partial element that contains a collection of components that make up a partial expression.	Section 4.1.5
stream	0	Unlimited	Stream element that defines the output of a particular receive stream. There should only be one stream element per receive stream supported by the card.	Section 4.1.6

4.1.2 <filter>

This element defines the construction of a filter, multiple filters can be defined but only one filter element per filter number is allowed.

Table 4-4. <filter> Attributes

Attribute	Use	Type	Default Value	Description
enabled	Optional	boolean	true	Defines the filter as either enabled or disabled. By default the filter is enabled. Possible values for this attribute are `true` or `false`.

Table 4-5. <filter> Elements

Element	Occurrences		Description	Section
	Min	Max		
name	1	1	The name of the filter, this is a user defined name that is used when creating partial expressions.	-
number	1	1	The number of the filter, this should be value in the range of 0-6. It is an error to have two filters with the same number.	-
sonet	0	1	Defines the filter as a sonet filter, the filter will be created with the assumption that packets being received are encapsulated in PoS fames.	Section 4.1.8
ethernet	0	1	Defines the filter as an ethernet filter, the filter is constructed with the assumption that packets will be encapsulated inside an ethernet frame.	Section 4.1.9
ethernet-vlan	0	1	Defines the filter as an ethernet with VLAN tags, the filter is constructed with the assumption that packets will be encapsulated inside a ethernet frame with a four byte VLAN tag.	Section 4.1.9
raw	0	1	Defines the filter as a raw filter, raw filters have the filter bits defined directly in the xml file.	Section 4.1.7
early-term	0	1	Defines the early termination length, this is the last filter element (8 byte chunk) of the filter that will be used to compare against the packet. If this element is not specified it defaults to the maximum size of the filter (currently 8 elements). The packet record is compared up to the last non-masked byte of the element, packets that are smaller than the early termination length are dropped. Example: `<early-term>1</early-term>` Refer to section 2.1.1 for more information on the early termination option.	-

4.1.3 <interface>

Defines an interface that can be used inside a partial expression.

Table 4-6. <interface> Elements

Element	Occurrences		Description	Section
	Min	Max		
name	1	1	The name of the interface, this is a user defined name that is used when creating partial expressions.	-
number	1	1	The number of the interface, this should be value in the range of 0-3.	-

4.1.4 <steering>

Defines a steering algorithm that can be used inside a partial expression.

Table 4-7. <steering> Elements

Element	Occurrences		Description	Section
	Min	Max		
name	1	1	The name of the interface, this is a user defined name that is used when creating partial expressions.	-

algorithm	1	1	Defines the algorithm used for steering. The two possible values are `parity` and `crc32`.
			
			Examples:
			`<steering>`
			`<name>steer0</name>`
			`<algorithm>parity</algorithm>`
			`</steering>`
			
			`<steering>`
			`<name>steer1</name>`
			`<algorithm>crc32</algorithm>`
			`</steering>`
				-

4.1.5 <partial>

Defines a partial expression, this element is expected to contain one or more `<partial-component>` elements, each defining either a filter, interface or steering algorithm to OR together to create a complete partial expression. For example the following XML snippet will produce:

Example 1:

```
partial0 = filter0 OR NOT filter1 OR interface0
```

```
<partial>
       <name>partial0</name>
       <partial-component>filter0</partial-component>
       <partial-component invert="true">filter1</partial-component>
       <partial-component>interface0</partial-component>
</partial>
```

Table 4-8. <steering> Elements

Element	Occurrences		Description	Section
	Min	Max		
name	1	1	The name of the partial expression, this is a user defined name that is used when creating a stream output expressions.	-
partial-component	0	Unlimited	Specifies the name of a partial component that makes up the partial expression.	Section 4.1.21

4.1.6 <stream>

This element specifies the partial expressions that are AND'ed together to create a complete stream output expression.

If a stream element is not specified for a particular stream number, that stream is effectively disabled, no packets regardless of filters, steering algorithms or interface will ever be sent to that stream.

Table 4-9. <stream> Elements

Element	Occurrences		Description	Section
	Min	Max		
number	1	1	The receive stream number, this must be an even decimal number between 0 and the maximum number of receive streams.	-
stream-component	1	Unlimited	Specifies the name of a partial expression that makes up the stream output expression.	Section 4.1.22

Note: packets can't be routed to multiple streams, therefore if the result of the filters, steering algorithm, interface and output expression applied by the DSM produces multiple stream outputs, the DSM library will adjust the output expression automatically to allow only the highest priority stream to receive the packet. The lower the stream number the higher the priority.

The following examples illustrate simple stream output expressions.

Example 1:

```
stream0 = partial0 AND partial1 AND NOT partial2
```

```
<stream>
       <number>0</number>
       <stream-component>partial0</stream-component>
       <stream-component>partial1</stream-component>
       <stream-component invert="true">partial2</stream-component>
</stream>
```

Example 2:

```
stream2 = (filter0 OR filter6) AND NOT (interface0 OR NOT steering0 OR
filter1)


<partial>
       <name>partial0</name>
       <partial-component>filter0</partial-component>
       <partial-component>filter6</partial-component>
</partial>

<partial>
       <name>partial1</name>
       <partial-component>interface0</partial-component>
       <partial-component invert="true">steering0</partial-component>
       <partial-component>filter1</partial-component>
</partial>

<stream>
       <number>2</number>
       <stream-component>partial0</stream-component>
       <stream-component invert="true">partial1</stream-component>
</stream>
```

Example 3:

It is possible to construct a stream output expression that will never accept, as in the following example:

```
stream0 =  (filter0) AND NOT (filter0)

<partial>
       <name>partial0</name>
       <partial-component>filter0</partial-component>
</partial>

<stream>
       <number>0</number>
       <stream-component>partial0</stream-component>
       <stream-component invert="true">partial0</stream-component>
</stream>
```

The `dsm_loader` program doesn't check for such situations, it is the user responsibility to supply correctly structured output expressions.

4.1.7 <raw>

This element defines the parent filter as being of type raw. This element

can only have child elements of type <word>, each word represents 32-bits of the filter, they are parsed in top down order with the first word element corresponding to the first 32-bits of the filter.

Table 4-10. <raw> Elements

Element	Occurrences		Description	Section
	Min	Max		
word	0	16	Specifies the 32-bits to filter on, the characters in the contents of the element must be either '0', '1', or '-' for don't care.	-

It is not necessary to define all 16 words that make a complete filter, instead words that aren't specified are assumed to be don't cares, the following example illustrates three filters that are identical:

Example 1:

```
<filter>
       <name>filter0</name>
       <raw>
              <word>10101010101010101010101010----</word>
              <word>1010------10101010101010101010</word>
              <word>--------------------------------</word>
              <word>--------------------------------</word>
              <word>--------------------------------</word>
              <word>--------------------------------</word>
              <word>--------------------------------</word>
              <word>--------------------------------</word>
              <word>--------------------------------</word>
              <word>--------------------------------</word>
              <word>--------------------------------</word>
       </raw>
</filter>
<filter>
       <name>filter1</name>
       <raw>
              <word>10101010101010101010101010----</word>
              <word>1010------10101010101010101010</word>
              <word>--------------------------------</word>
              <word>--------------------------------</word>
       </raw>
</filter>
<filter>
       <name>filter2</name>
       <raw>
              <word>10101010101010101010101010----</word>
              <word>1010------10101010101010101010</word>
       </raw>
</filter>
```

4.1.8 <sonet>

This element defines the parent filter element as a sonet type filter, sonet filters expect PoS packets and therefore adjust the position of the layer3 and layer4 filter elements accordingly. Defining a filter as being of type sonet, doesn't force the filter to be that type, if the DSM library detects that the card being configured is not sonet (for example ethernet instead) the sonet specific child elements are ignored and the library automatically compensates for the correct layer 2 encapsulation.

Table 4-11. <sonet> Elements

Element	Occurrences		Description	Section
	Min	Max		
hdlc-type	0	1	Defines the PoS (HDLC/PPP) header to filter on.	Section 4.1.10
ipv4	0	1	Defines the IPv4 fields to filter on.	Section 4.1.13

2.1.9 <ethernet> & <ethernet-vlan>

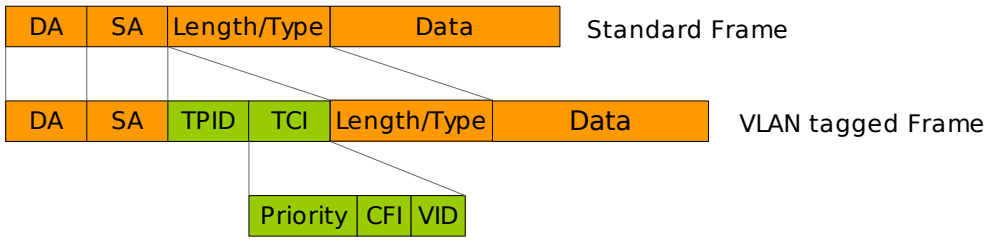
These elements defines the parent filter as a ethernet type filter (with or without VLAN), ethernet filters expect packets to be encapsulated in ethernet frames and therefore adjust the position of the layer3 and layer4 filter elements accordingly. As with the sonet element defining a filter as being an ethernet type doesn't force the filter to be that type, if the DSM library detects that the card being configured is not ethernet (for example sonet instead) the ethernet specific child elements are ignored and the library automatically compensates for the correct layer 2 encapsulation.

Table 4-12. <ethernet> & <ethernet-vlan> Elements

Element	Occurrences		Description	Section
	Min	Max		
ethertype	0	1	Defines the 2 byte ethertype value to filter on.	Section 4.1.11
mac-source	0	1	Defines the source ethernet MAC address to filter on.	Section 4.1.12
mac-dest	0	1	Defines the destination ethernet MAC address to filter on.	Section 4.1.12
ipv4	0	1	Defines the IPv4 fields to filter on.	Section 4.1.13
vlan-id	0	1	Defines the 12-bit VLAN Id to filter on, this element is ignored if the `<ethernet>` element is the parent.	Section 4.1.23

When `<ethernet-vlan>` is used, the DSM library will automatically add an entry in the filter for the VLAN ethertype (0x8100) and the user defined ethertype (if specified by the `<ethertype>` element) is relocated to the lower 16-bits of the VLAN tag as per the IEEE 802.1Q standard.

Figure 4-1 IEEE 802.1Q Standard, Tag-based VLAN



4.1.10 <hdlc-type>

The contents of this element define the hdlc header to filter on, this is a 32-bit value. There is no mask for this element so the filter will hit on a direct match only.

Table 4-13. <hdlc-type> Attributes

Attribute	*Use*	*Type*	*Default Value*	*Description*
hex	Optional	boolean	false	Indicates whether the contents of the element are in hexadecimal form or the default decimal form.

4.1.11 <ethertype>

The contents of this element define the ethertype value to filter on, this is a 16-bit value. There is no mask for this element so the filter will hit on a direct match only.

Table 4-14. <hdlc-type> Attributes

Attribute	*Use*	*Type*	*Default Value*	*Description*
hex	Optional	boolean	false	Indicates whether the contents of the element are in hexadecimal form or the default decimal form.

4.1.12 <mac-source> & <mac-dest>

These elements defines the source/destination MAC address to filter on. If the `<addr>` child element is not present the address defaults to all zeros, if the `<mask>` child element is not present the mask defaults to all ones.

Table 2-15. <mac-source> & <mac-dest> Elements

Element	Occurrences		Description	Section
	Min	Max		

addr	0	1	Defines the MAC address to filter on, the format of the contents should be `X:X:X:X:X:X` where `X` is either a decimal number (0-255) or a hexadecimal number (0-FF). The `hex` attribute determines the format of the address, by default decimal is used. Add `hex="true"` to the element to enable hex parsing.
			
			Examples:
			`<addr>12:34:56:78:90:12</addr>`
			
			`<addr hex="true">`
			`99:AA:BB:CC:DD:EE:FF`
			`</addr>`
				-

mask	0	1	Defines the mask to use for the MAC address, the contents have the same format as the `<addr>` element.	-

4.1.13 <ipv4>

This element encapsulates the IPv4 specific filter elements. By defining this element it doesn't automatically change the ethertype/hdlc-type of the filter to match IPv4. If this is required you can do it manually by defining a `<ethertype>` or `<hdlc-type>` element inside the parent element.

Table 4-16. <ipv4> Elements

Element	Occurrences		Description	Section
	Min	Max		
ip-source	0	1	Defines the IPv4 source address to filter on.	Section 4.1.14
ip-dest	0	1	Defines the IPv4 destination address to filter on.	Section 4.1.14
no-ip-frags	0	1	If present, specifies that the filter will filter out all IPv4 packets that have been fragmented. If this element is not present the filter ignores whether or not the packet is a fragment.	-
ihl	0	1	Defines the IP header length to filter on, this element doesn't contain a mask therefore a direct match is required for a hit. The `hex` attribute can be specified if the contents of the element is in hexadecimal. The `hex` attribute defaults to `"false"`. As with the IHL field in the IPv4 header this element should contain the length of the header in 32-bit words, for example `<ihl>6</ihl>` would define a header length of 24 bytes.	-

			The minimum header length is 5 and the maximum is 15. This element also effects the offset of any subsequent layer4 (UDP, TCP & ICMP) details,	
tcp	0	1	Defines the layer4 type as TCP.	Section 4.1.15
udp	0	1	Defines the layer4 type as UDP.	Section 4.1.15
icmp	0	1	Defines the layer4 type as ICMP.	Section 4.1.18

4.1.14 <ip-source> & <ip-dest>

These elements define the IPv4 source/destination addresses to filter on. If the `<addr>` child element is not present the address defaults to all zeros, if the `<mask>` child element is not present the mask defaults to all ones.

Table 4-17. <ip-source> & <ip-dest> Elements

Element	*Occurrences*		*Description*	*Section*
	Min	*Max*		

addr	0	1	Defines the IPv4 address to filter on, the contents should be formatted like `X.X.X.X` where `X` is either a decimal number (0-255) or a hexadecimal number (0-FF). The `hex` attribute determines the format of the address, by default decimal is used. Add `hex="true"` to the element to enable hexadecimal parsing.
			
			Examples:
			`<ip-source>`
			`<addr>192.168.0.0</addr>`
			`</ip-source>`
			
			`<ip-dest>`
			`<addr hex="true">C0.A8.0.0</addr>`
			`<mask hex="true">FF.FF.0.0</mask>`
			`</ip-dest>`
				-

mask	0	1	Defines the mask to use for the IPv4 address, the contents have the same format as the `<addr>` element.	-

4.1.15 <tcp> & <udp>

These elements defines the layer4 type as either UDP or TCP. The protocol field in the IPv4 header is set to either `0x06` (TCP) or `0x11` (UDP) and the filter is updated.

Table 4-18. <tcp> & <udp> Elements

Element	Occurrences		Description	Section
	Min	Max		
source-port	0	1	Defines the UDP/TCP source port to filter on.	Section 4.1.16
dest-port	0	1	Defines the UDP/TCP destination port to filter on.	Section 4.1.16
tcp-flags	0	1	Defines the TCP flags to filter on, this element is only valid as the child of a <tcp> element,	Section 4.1.17

4.1.16 <source-port> & <dest-port>

These elements define the UDP/TCP source/destination ports to filter on. If the `<port>` child element is not present the port defaults to all zeros, if the `<mask>` child element is not present the mask defaults to all ones.

Table 4-19. <source-port> & <dest-port> Elements

Element	Occurrences		Description	Section
	Min	Max		

port	0	1	Defines the UDP/TCP port to filter on, the format of the contents should be either a decimal number (0-65535) or a hexadecimal number (0-FFFF). The `hex` attribute determines the format of the port, by default decimal is used. Add `hex="true"` to the element to enable hexadecimal port values.
			
			Examples:
			`<dest-port>`
			`<port>80</port>`
			`<mask>255</mask>`
			`</dest-port>`
			
			`<source-port>`
			`<port hex="true">00AF</port>`
			`<mask hex="false">255</mask>`
			`</source-port>`
				-

mask	0	1	Defines the mask to use for the UDP/TCP port, the contents have the same format as the `<port>` element.	-

4.1.17 <tcp-flags>

This element defines the TCP flags to filter on. If the `<flags>` element is not present the flags defaults to all zeros, if the `<mask>` element is not present the mask defaults to all ones.

Table 4-20. <tcp-flags> Elements

Element	Occurrences		Description	Section
	Min	Max		

flags	0	1	Defines the TCP flags to filter on, the format of the contents should be either a decimal number (0-63) or a hexadecimal number (0-3F). The `hex` attribute determines the format of the flags, by default decimal is used. Add `hex="true"` to the element to enable hexadecimal port values.
			
			Examples:
			`<flags>31</flags>`
			
			`< flags hex="true">1F</flags>`
				-

mask	0	1	Defines the mask to use for the TCP flags, the contents have the same format as the `<flags>` element.	-

4.1.18 <icmp>

This element defines the layer4 type as ICMP. The protocol field in the IPv4 header is set to `0x01` (the ICMP protocol number) and the filter is updated.

Table 4-21. <icmp> Elements

Element	Occurrences		Description	Section
	Min	Max		
icmp-code	0	1	Defines the ICMP code to filter on.	Section 4.1.19
icmp-type	0	1	Defines the ICMP type to filter on.	Section 4.1.20

4.1.19 <icmp-code>

This element defines the ICMP code to filter on. If the `<code>` element is not present the code defaults to all zeros, if the `<mask>` element is not present the mask defaults to all ones.

Table 4-22. <icmp-code> Elements

Element	Occurrences		Description	Section
	Min	Max		
code	0	1	Defines the ICMP code to filter on, the format of the contents should be either a decimal number (0-255) or a hexadecimal number (0-FF). The `hex` attribute determines the format of the code, by default decimal is used. Add	-

			hex="true" to the element to enable hexadecimal code values. Examples: <icmp-code> <code>31</code> <mask>255</mask> </icmp-code> <icmp-code> <code hex="true">1F</code> <mask hex="true">FF</mask> </icmp-code>	
mask	0	1	Defines the mask to use for the ICMP code, the contents have the same format as the <code> element.	-

4.1.20 <icmp-type>

This element defines the ICMP type to filter on. If the <type> element is not present the type defaults to all zeros, if the <mask> element is not present the mask defaults to all ones.

Table 4-23. <icmp-type> Elements

Element	*Occurrences*		*Description*	*Section*
	Min	*Max*		
type	0	1	Defines the ICMP type to filter on, the format of the contents should be either a decimal number (0-255) or a hexadecimal number (0-FF). The hex attribute determines the format of the type, by default decimal is used. Add hex="true" to the element to enable hexadecimal type values. Examples: <icmp-type> <type>31</type> <mask>255</mask> </icmp-type> <icmp-type> <type hex="true">1F</type> <mask hex="true">FF</mask> </icmp-type>	-
mask	0	1	Defines the mask to use for the ICMP type, the contents have the same format as the <type> element.	-

4.1.21 <partial-component>

Defines either a filter, interface or steering algorithm that is part of a partial expression, each component is OR'ed together to create a partial expression. The component must correspond to a named `<filter>`, `<interface>` or `<steering>` element in the file.

Table 4-24. <partial-component> Attributes

Attribute	Use	Type	Default Value	Description
invert	Optional	boolean	false	Indicates that the inversion of the component should be used in the partial expression.

Note: Although it is possible to specify a component and it's inverse in a partial expression, the actual outcome is that the last component is the one that is set. For example in the following xml snippet:

```
<partial>
       <name>partial0</name>
       <partial-component>filter0</partial-component>
       <partial-component invert="true">filter0</partial-component>
</partial>
```

you would expect the output to be `partial0 = filter0 OR NOT filter0` (an accept all expression), instead simply `partial0 = NOT filter0` will be produced, because the second `<partial-component>` element was processed last.

4.1.22 <stream-component>

Defines a partial expression that is part of a stream output expression, each partial expression is AND'ed together to form an output expression. The component must correspond to a named `<partial>` child element in the file.

Table 4-25. <stream-component> Attributes

Attribute	Use	Type	Default Value	Description
invert	Optional	boolean	false	Indicates that the inversion of the partial should be used in the stream output expression.

4.1.23 <vlan-id>

This element defines the VLAN Id to filter on this is the 12-bit id contained within the VLAN tag of a packet. If the `<id>` child element is not present the ID defaults to all zeros, if the `<mask>` child element is not

present the mask defaults to all ones.

Table 4-26. <vlan-id> Elements

Element	Occurrences		Description	Section
	Min	Max		

id	0	1	Defines the 12-bit VLAN ID to filter on, the format of the contents should be either a decimal number (0-4095) or a hexadecimal number (0-FFF). The `hex` attribute determines the format of the ID, by default decimal is used. Add `hex="true"` as an attribute to the element to enable hexadecimal ID values.
			
			Examples:
			`<vlan-id>`
			`<id>31</id>`
			`<mask>63</mask>`
			`</vlan-id>`
			
			`<vlan-id>`
			`<id hex="true">1F</id>`
			`<mask hex="true">3F</mask>`
			`</vlan-id>`
				-

mask	0	1	Defines the mask to use for the VLAN ID, the contents have the same format as the <id> element.	-

4.2 Example Filter Files

This section provides example DSM filter files.

4.2.1 Simple Steering

The following example uses the crc32 steering algorithm to approximately split the received packets into two streams (streams 0 and 2). The physical port and filters are ignored.

stream0 = steer0

stream2 = **NOT** steer0

Example file:

```
<?xml version="1.0"?>

<dsm-config version="1.0">
	<steering>
		<name>steer0</name>
		<algorithm>crc32</algorithm>
	</steering>

	<partial>
		<name>partial0</name>
		<partial-component>steer0</partial-component>
	</partial>

	<stream>
		<number>0</number>
		<stream-component>partial0</stream-component>
	</stream>
	<stream>
		<number>2</number>
		<stream-component invert="true">partial0</stream-component>
	</stream>
</dsm-config>
```

4.2.2 Interface Steering

This example assumes that the configuration will be loaded into a DAG card that has four phyiscal ports (interfaces). The configuration routes all traffic that arrives on ports 0 & 2 to receive stream 0, traffic received on port 1 is routed to stream 2 and any traffic that arrives on port 3 is dropped.

The outputs of both steering algorithms and the eight filters are ignored.

stream0 = iface0 **OR** iface2

stream2 = iface1

Example file:

```
<?xml version=”1.0”?>
```

```
<dsm-config version=”1.0”>
	<interface>
		<name>iface0</name>
		<number>0</number>
	</interface>
	<interface>
		<name>iface1</name>
		<number>1</number>
	</interface>
	<interface>
		<name>iface2</name>
		<number>2</number>
	</interface>

	<partial>
		<name>partial0<name>
		<partial-component>iface0</partial-component>
		<partial-component>iface2</partial-component>
	</partial>
	<partial>
		<name>partial1<name>
		<partial-component>iface1</partial-component>
	</partial>

	<stream>
		<number>0</number>
		<stream-component>partial0</stream-component>
	</stream>
	<stream>
		<number>2</number>
		<stream-component>partial1</stream-component>
	<stream>
</dsm-config>
```

4.2.3 Simple Filtering

The following example routes all IPv4 packets that have a class B source IP address of 192.168.x.x to stream 0, all other TCP packets with a destination port of 80 are routed to stream 2 and everything else is dropped. The DAG card is assumed to be ethernet without VLAN.

Example file:

```
<?xml version="1.0"?>

<dsm-config version="1.0">

	<filter>
		<name>filter0</name>
		<number>0</number>
		<ethernet>
			<ipv4>
				<ip-source>
					<addr>192.168.0.0</addr>
					<mask hex="true">FF.FF.0.0</mask>
				</ip-source>
				<ip-dest>
					<addr hex="true">0.0.0.0</addr>
					<mask hex="true">0.0.0.0</mask>
				</ip-dest>
			</ipv4>
		</ethernet>
	</filter>
```

```
        <filter early-term="false">
               <name>filter1</name>
               <number>1</number>
               <ethernet>
                      <ipv4>
                             <tcp>
                                    <dest-port>
                                           <port>80</port>
                                           <mask hex="true">FFFF</mask>
                                    </dest-port>
                             </tcp>
                      </ipv4>
               </ethernet>
        </filter>


        <partial>
               <name>partial0</name>
               <partial-component>filter0</partial-component>
        </partial>
        <partial>
               <name>partial1</name>
               <partial-component>filter1</partial-component>
        </partial>

        <stream>
               <number>0</number>
               <stream-component>partial0</stream-component>
        </stream>
        <stream>
               <number>2</number>
               <stream-component>partial1</stream-component>
        </stream>

</dsm-config>
```

4.2.4 Complete Example

This example demonstrates how to create a complete example that utilises the filter, interface and steering algorithms to route the packets between the two receive streams.

stream0 = (filter0 **OR** filter1) **AND NOT** (iface0 **OR** iface2) **AND** (steer0)

stream2 = (filter2 **OR** filter3) **AND NOT** (iface1 **OR** iface3) **AND** (steer1)

Example file:

```
<?xml version="1.0"?>

<dsm-config version="1.0">

        <!-- ethernet IPv4 filter for odd source addresses -->
        <filter>
               <name>filter0</name>
               <number>0</number>
               <ethernet>
                      <ipv4>
                             <ip-source>
                                    <addr>0.0.0.1</addr>
```

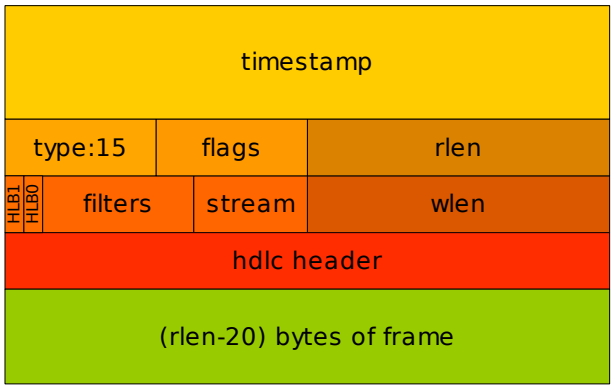
```
                        <mask>0.0.0.1</mask>
                    </ip-source>
                </ipv4>
        </ethernet>
</filter>

<!-- ethernet IPv4 filter for odd destination addresses -->
<filter>
        <name>filter1</name>
        <number>1</number>
        <ethernet>
                <ipv4>
                    <ip-dest>
                        <addr>0.0.0.1</addr>
                        <mask>0.0.0.1</mask>
                    </ip-dest>
                </ipv4>
        </ethernet>
</filter>

<!-- ethernet IPv4 filter for ICMP ping requests -->
<filter>
        <name>filter2</name>
        <number>2</number>
        <ethernet>
                <ipv4>
                    <icmp>
                        <icmp-type>
                            <type>8</type>
                            <mask hex="true">FF</mask>
                        </icmp-type>
                    </icmp>
                </ipv4>
        </ethernet>
</filter>

<!-- ethernet IPv4 filter for ICMP ping replies -->
<filter>
        <name>filter3</name>
        <number>3</number>
        <ethernet>
                <ipv4>
                    <icmp>
                        <icmp-type>
                            <type>0</type>
                            <mask hex="true">FF</mask>
                        </icmp-type>
                    </icmp>
                </ipv4>
        </ethernet>
</filter>

<!-- Interfaces -->
<interface>
        <name>iface0</name>
        <number>0</number>
</interface>
<interface>
        <name>iface1</name>
        <number>1</number>
</interface>
<interface>
        <name>iface2</name>
        <number>2</number>
```

```
        </interface>
        <interface>
               <name>iface3</name>
               <number>3</number>
        </interface>

        <!-- Steering algorithms -->
        <steering>
               <name>steer0</name>
               <algorithm>crc32</algorithm>
        </steering>
        <steering>
               <name>steer1</name>
               <algorithm>parity</algorithm>
        </steering>

        <!-- Partial expressions -->
        <partial>
               <name>partial0</name>
               <partial-component>filter0</partial-component>
               <partial-component>filter1</partial-component>
        </partial>
        <partial>
               <name>partial1</name>
               <partial-component>filter2</partial-component>
               <partial-component>filter3</partial-component>
        </partial>
        <partial>
               <name>partial2</name>
               <partial-component>iface0</partial-component>
               <partial-component>iface2</partial-component>
        </partial>
        <partial>
               <name>partial3</name>
               <partial-component>iface1</partial-component>
               <partial-component>iface3</partial-component>
        </partial>
        <partial>
               <name>partial4</name>
               <partial-component>steer0</partial-component>
        </partial>
        <partial>
               <name>partial5</name>
               <partial-component>steer1</partial-component>
        </partial>

        <!-- Output (stream) expressions -->
        <stream>
               <number>0</number>
               <stream-component>partial0</stream-component>
               <stream-component invert="true">partial2</stream-component>
               <stream-component>partial4</stream-component>
        </stream>
        <stream>
               <number>2</number>
               <stream-component>partial1</stream-component>
               <stream-component invert="true">partial3</stream-component>
               <stream-component>partial5</stream-component>
        </stream>

</dsm-config>
```

Appendix A – ERF Record Format

The following ERF record formats are generated by the DSM module, refer to the *EDM11-01 Endace Extensible Record Format* document for more detailed information.

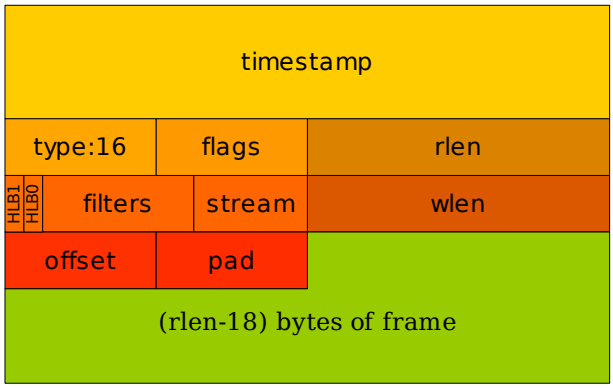
A.1 Type 15 DSM Colored PoS HDLC Record



Data Format	Size	Description
timestamp	64 bits	The time of arrival of the cell, an ERF 64-bit timestamp, described in more detail in the *EDM11-01* document.
type	8 bits	This field contains an enumeration of the frame subtype. For DSM colored PoS records this value should be 15.
flags	8 bits	This byte is divided into 2 parts, the interface identifier, and a set of 1-bit flags.
rlen	16 bits	Record length. Total length of the record transferred over PCI bus to storage.
hlb1 & hlb0	1 bit each	Contains the output of each of the load balancing algorithms. `hlb0` = CRC algorithm output. `hlb1` = Parity algorithm output.
filters	8 bits	Each bit represents an output from one of the filters. `1` indicates a `true` output, `0` indicates a `false` output. Filter `0` is bit `0` of this field.
stream	6 bits	The target receive stream for the packet record.
wlen	16 bits	Wire length. The length of the packet that was received from the line.
hdlc header	32 bits	The 32-bit PoS frame HDLC header that was

		received from the line.

A.2 Type 16 DSM Colored Ethernet Record



Data Format	*Size*	*Description*
timestamp	64 bits	The time of arrival of the cell, an ERF 64-bit timestamp, described in more detail in the *EDM11-01* document.
Type	8 bits	This field contains an enumeration of the frame subtype. For DSM colored Etherent records this value should be 16.
flags	8 bits	This byte is divided into 2 parts, the interface identifier, and a set of 1-bit flags.
rlen	16 bits	Record length. Total length of the record transferred over PCI bus to storage.
hlb1 & hlb0	1 bit each	Contains the output of each of the load balancing algorithms. `hlb0` = CRC algorithm output. `hlb1` = Parity algorithm output.
filters	8 bits	Each bit represents an output from one of the filters. `1` indicates a `true` output, `0` indicates a `false` output. Filter `0` is bit `0` of this field.
stream	6 bits	The target receive stream for the packet record.
wlen	16 bits	Wire length. The length of the packet that was received on the line.
offset	8 bits	This field is currently not implemented, contents can be disregarded.
pad	8 bits	Padding byte so the ethernet frame is aligned on a 16-bit boundary.

Appendix B – Raw DSM Configuration Output

This appendix describes the format of the `dsm_loader` generated raw configuration output files. The output files are generated when the `-o` option is used with `dsm_loader`.

The files are divided into two sections, the first containing the seven filter configurations, the second containing the lookup table.

B.1 Filters

The first line of the filter entry contains the attributes of the filter, possible values are given in the following table.

Table B-1. Filter Attributes

Name	*Description*
`soft-filter`	Indicates the filter is derived from a software configuration, rather than directly from the DAG card. This option is always specified.
`virtual filter actual filter`	Defines the virtual filter number and the actual filter number. The virtual filter number is the value used in the virtual configuration, whereas the actual filter number is the number of the filter inside the DAG card.
`enabled`	Indicates the filter is enabled
`disabled`	Indicates the filter is disabled
`Early-terminate on element`	Indicates the first element that has the early termination option set.

Following the attributes are sixteen lines of raw filter data, the first column contains the number of the filter, the next is the address of the filter data relative to the start of the filter, following that are 32 characters displaying the raw filter bits. A '`-`' signifies that bit is a 'don't care' value, '`0`' and '`1`' indicate bits that should be matched by the filter. The last two columns contain the hexadecimal value and mask words of the filter.

If a `*` is present at the end of the line it indicates that the early termination option has been set on that element, the early termination option is set over an element which is 64-bits in size, therefore the `*` will always be shown on two lines. By default the early termination option is always set on the last element of the filter if it hasn't been user defined.

Example B-1. DSM filter output

```
soft-filter, virtual filter 2, actual filter 2, enabled, early-terminate on element 2
```

```
Filter2 : 00 [----------------1010101010111011]  Value[0000AABB] Mask[0000FFFF]
Filter2 : 04 [11001100110111011111011011111111]  Value[CCDDEEFF] Mask[FFFFFFFF]
Filter2 : 08 [00010010001101000101011001111000]  Value[12345678] Mask[FFFFFFFF]
Filter2 : 0C [10010000000100101000000100000000]  Value[90128100] Mask[FFFFFFFF]
Filter2 : 10 [----00000001001001000010001000100]  Value[00124444] Mask[0FFFFFFF] *
Filter2 : 14 [--------------------------------]  Value[00000000] Mask[00000000] *
Filter2 : 18 [--------------------------------]  Value[00000000] Mask[00000000]
Filter2 : 1C [--------00000001----------------]  Value[00010000] Mask[00FF0000]
Filter2 : 20 [1100000010101000----------------]  Value[C0A80000] Mask[FFFF0000]
Filter2 : 24 [--------1010100000010111--------]  Value[00A81700] Mask[00FFFF00]
Filter2 : 28 [1000000001010000----------------]  Value[80500000] Mask[FFFF0000]
Filter2 : 2C [--------------------------------]  Value[00000000] Mask[00000000]
Filter2 : 30 [--------------------------------]  Value[00000000] Mask[00000000]
Filter2 : 34 [--------------------------------]  Value[00000000] Mask[00000000]
Filter2 : 38 [--------------------------------]  Value[00000000] Mask[00000000]
Filter2 : 3C [--------------------------------]  Value[00000000] Mask[00000000]
```

B.2 Lookup Table

The first four lines of the lookup table output, contain the statistics for the table. The `Bits per Entry` field, indicates how many bits are used to encoded the output of each entry, this is calculated automatically by the DSM API using the following formulae $\texttt{Bits per Entry} = \log_2(n) + 1$, where `n` is the number of receive memory holes. The `entries` field shows how many entries there are in the lookup table. The `Entries per Row` indicates the number of entries that can fit within a single row (16-bits), and `Rows` are the number of rows required to populate the complete table. The `Rows` referred to here is not related to a line of characters in the output file, rather it is a term used internally by the DSM API.

Each line of the output matches to a single entry in the lookup table, the `hlb-par`, `hlb-crc`, `iface` & `f7-f0` columns refer to the `true`/`false` input parameters into the lookup table. The `bm` column indicates either the output stream number or `drop`. The `row entry` column is the raw data that is programmed into the DAG card.

Example B-2. DSM Lookup table output

```
 Bits per Entry: 2
        Entries: 4096
Entries Per Row: 8
           Rows: 512

       : hlb-par hlb-crc  iface  f7 f6 f5 f4 f3 f2 f1 f0 : bm   | row entry
-------------------------------------------------------------------------------
0x000  :    0       0      00     0  0  0  0  0  0  0  0 : drop |  0xAAAA
0x001  :    0       0      00     0  0  0  0  0  0  0  1 : drop |  0xAAAA
0x002  :    0       0      00     0  0  0  0  0  0  1  0 : drop |  0xAAAA
0x003  :    0       0      00     0  0  0  0  0  0  1  1 : drop |  0xAAAA
0x004  :    0       0      00     0  0  0  0  0  1  0  0 : drop |  0xAAAA
0x005  :    0       0      00     0  0  0  0  0  1  0  1 : drop |  0xAAAA
...
...
...
0xFFC  :    1       1      11     1  1  1  1  1  1  0  0 : drop |  0x0202
0xFFD  :    1       1      11     1  1  1  1  1  1  0  1 :  0   |  0x0202
0xFFE  :    1       1      11     1  1  1  1  1  1  1  0 :  0   |  0x0202
0xFFF  :    1       1      11     1  1  1  1  1  1  1  1 :  0   |  0x0202
```